12-2015

Empirical Study of Training-Set Creation for Software Architecture Traceability Methods

Waleed Abdu Zogaan
waz7355@rit.edu

Empirical Study of Training-Set Creation for Software Architecture Traceability Methods

by

WALEED ABDU ZOGAAN

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science
in
Software Engineering

Supervised by

Dr. Mehdi Mirakhorli

Department of Software Engineering

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

December 2015

The thesis “Empirical Study of Training-Set Creation for Software Architecture Traceability Method” by WALEED ABDU ZOGAAN has been examined and approved by the following Examination Committee:

Dr. Mehdi Mirakhorli
Assistant Professor
Thesis Committee Chair

Dr. Scott Hawker
Associate Professor

Dr. Stephanie Ludi
Professor

To the spirit of my beloved father, who I was hoping to see this day.

To my mother, whose without her love and prayers, I could not accomplished this.

To Wail, Wesam, Reham and Rawan, for their endless love, support and encouragement.

To my wife, Hend, whose love and endless support is a constant source of energy.

And to my kids, Emaa, Khalid, Abdu and Mohab, whose smiles bright up my life.

Acknowledgments

I wish to thank my advisor, Dr. Mehdi Mirakhorli, for his endless guidance and support during my Master journey. Also, I am deeply thankful to my committee member, Dr. Scott Hawker, for providing me with his expertise.

Also, I would like to thank my team members, Ibrahim Mujhid and Joanna Cecilia, for their support during the conduction of the experimental work.

Abstract

Empirical Study of Training-Set Creation for Software Architecture Traceability Methods

WALEED ABDU ZOGAAN

Supervising Professor: Dr. Mehdi Mirakhorli

Machine-learning algorithms have the potential to support trace retrieval methods making significant reductions in costs and human-involvement required for the creation and maintenance of traceability links between system requirements, system architecture, and the source code. These algorithms can be trained how to detect the relevant architecture and can then be sent to find it on its own. However, the long-term reductions in cost and effort face a significant upfront cost in the initial training of the algorithm. This cost comes in the form of needing to create training sets of code, which train the algorithm how to identify traceability links. These supervised or semi-supervised training methods require the involvement of highly trained, and thus expensive, experts to collect, and format, these data-sets. In this thesis, three baseline methods training datasets creation are presented. These methods are (i) Manual Expert-based, which involves a human-compiled dataset, (ii) Automated Web-Mining, which creates training datasets by collecting and data-mining APIs (specifically from technical-programming websites), and (iii) Automated Big-Data Analysis, which data-mines ultra-large code repositories to generate the training datasets. The trace-link creation accuracy achieved using each of these three methods is compared, and the cost/benefit comparisons between them is discussed. Furthermore, in a related area, potential correlations between training set size and the accuracy of recovering trace links is investigated. The results of this area of study indicate that the automated techniques, capable of creating very large training sets, allow for sufficient reliability in the problem of tracing architectural tactics. This indicates that these automated methods have potential applications in other areas of software traceability.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The structure and design of software architecture is the foundational aspect of all non-functional requirements. For any given piece of software, its primary purpose and function, whether that be word-processing, statistical analysis, database management, packet forwarding, entertainment or any of the myriad other uses to which computers are now put is only one part of the full array of considerations that go into the software's design. How secure, safe, and private is the software? How reliable is the software? What performance does the software offer or require? [7] These considerations and requirements that come from the answers to these questions mean that a software architect must consider an array of possible design solutions. These solutions will all have engineering trade-offs, meaning that the architect must evaluate the potential risk factors associated with the software's use, and select the solution that comes with the optimum tradeoff for the software requirements [7]. However, a number of specific architectural tactics for fulfilling requirements has grown over the years [6], representing accumulated institutional knowledge on which specific trade-offs represent optimum solutions for specific requirements and quality considerations, representing an array of solutions for various problems at a variety of scales [6, 7].

Tactics come in many different shapes and sizes [6, 7]. For example, reliability tactics

offer solutions for fault mitigation, detection, and recovery; performance tactics provide solutions for resource contention in order to optimize response time and throughput, and security tactics provide solutions for authorization, authentication, non-repudiation and other such factors.

However, all of the various artifacts of any software affect and should be traceable to each other. Tracking and tracing these relations between the software quality attributes, architectural tactics, design rationales and relevant areas of the code is a significant area of concern for software engineering. Such tracings between these factors can and will impact all areas of the software as it is updated and changed, affecting considerations at every level, from architecture-level change-impact analysis, requirement validation, design reasoning, safety-case construction, and long-term system maintenance [28, 32]. To give one example of such effects, experience has shown that, when developers make changes to the code without fully understanding and considering the underlying architectural decisions and associated quality concerns, the quality of the system architecture and the software qualities dependant on that architecture can and will be eroded [29, 32]. This erosion can be prevented or compensated for if trace links documenting the interdependencies of the software architecture are available, as the developers can then use that information to reduce the likelihood of undermining previous design decisions [30]. This is of particular importance in systems that are safety-critical, as modifications to such software can have adverse effects on the systems reliability, availability, dependability, and risk mitigation if such modifications are not handled with care [12].

Archie [27, 30, 32], an automated data-mining technique, presented by M. Mirakhorli et.al., is discussed in the Prior Work. This technique is used to trace software quality requirements through the architectural tactics to the source code. Archie includes a pre-rendered set of codebase classifiers constructed to detect various architectural tactics in the source code [27, 32]. Specifically, the individual classifiers have been trained to detect a variety of known software architecture tactics, including audit, asynchronous method invocation, authentication, checkpoint, heartbeat, role-based access control (RBAC), resource

pooling, scheduler, and secure session. Archie's classifiers were trained using open-source code snippets from hundreds of high-performance projects that contained these various architectural tactics.

As shown in the Prior Work, Archie's results were promising [27, 32], it was observed that the labor and time investment for extending this method to additional numbers of tactics would be cost-prohibitive, due to the sheer amount of highly-trained expert effort that would be required to create the new training sets. As a result, while effective, the technique cannot be scaled to industrial levels due to this cost, which has implications for industrial level software development in general.

However, the fact remains that the training sets for Archie were essentially manually-collected by highly experienced experts, who possessed deep understandings of how software quality attributes and architectural tactics are designed, implemented and interact. Furthermore, due to the skill requirements, attempting to use untrained or inexperienced coders (such as less-experienced developers or students) to create the data training set will be challenging, as they will lack the skills and knowledge required to search across the various systems, understand the code snippets, and select the snippets that are best-of-breed representatives of the tactics. Even when experts (e.g. architects) are involved in the process of creating such datasets, *the process is very time consuming* as it took a researchers three months to collect datasets for their experiments [31, 32].

Due to these cost barriers, present research into this area in the software development community is limited, typically using very small, student-created datasets, or limited industrial datasets that are unshareable due to copyright or non-disclosure agreements, which impacts the ability to engage in general, reproducible research.

1.1 Thesis Objectives

In this thesis we presents an empirical study on software traceability research, and novel techniques that advance the researchers previous work in this area in several important ways, and accomplish three objectives. First among these is presenting new approaches

based on (i) *Web-Mining* and (ii) *Big-Data Analysis* to automate the creation of traceability datasets. The Automated Web-Mining method creates training sets by mining APIs related to architecture tactics from websites focused on technical programming. In comparison, the Big-Data Analysis method uses analysis of ultra-large code repositories for the compilation of training sets; the code repository established for this thesis contains over 116,000 open source software projects.

The second objective is reporting a series of empirical studies conducted to compare the accuracy of a traceability technique trained using the automatically generated training-sets versus the datasets which are manually established by the experts.

Finally, the cost/benefit comparisons between the two automated approaches along with other application area of these approaches is empirically evaluated through a set of experiments.

1.2 Research Questions

In addition to these novel techniques, this thesis reports the results of empirical studies aimed at investigating the following research questions:

Research Question 1: Does the Training Method Based on Automated-Web Mining Result in Higher Trace-Links Classification Accuracy Compared to Expert Created Training Set?

Chapter 4 reports on the series of experiments conducted for this question. The empirical analysis indicates that the Web-Mining method presented here can and will produce high-quality training sets, with the accuracy of trace-link classifiers from the web-mining method being comparable with the expert-compiled dataset, with the differences being statistically insignificant. This result can potentially expand the present state of software architecture traceability, as it removed the economic barrier to the creation of training datasets via this technique.

Research Question 2: Does the Training Method Based on Automated Big-Data Analysis Result in Higher Trace-Links Classification Accuracy Compared to Expert Created Training Set?

The results from the empirical study indicate that the novel Big-Data Analysis method will create high-quality training data-sets. In three out of five of the empirical experiments performed, the accuracy of the trace-link classifier trained using data-sets created using this method was an improvement over the trace-link classifier trained on an expert-created dataset. Statistical analysis indicated that the difference between these two data set creation methods was not statistically significant in regards to the classifier outcomes. Therefore the Big-Data Analysis approach can be used to help researchers create high quality datasets of architectural code snippets.

Research Question 3: What is the Impact of Training Set Size on the Accuracy of Trace Link Classification?

The results of the empirical studies, as interpreted by a cost-benefit analysis, indicated that there is not a statistically significant degree of difference in accuracy for the trace-link classifier based on training set sizes. Classifiers were trained on training sets containing 10, 20, 30, 40, and 50 projects for this analysis. As such, expert-created training sets are potentially still viable.

Research Question 4: Can Automated Training-Set Creation Approaches Be Applied to the Regulatory Codes and Functional Requirements Traceability Scenarios?

The focus of this automated dataset generation technique was for creating training datasets in software architecture traceability. However, there is no direct reason why this

method cannot be used for other areas of software research. A feasibility study was conducted; while not the main focus of this thesis, this study does indicate that this area has potential for future work in these research areas.

1.3 Contribution

This work differs from prior publications by the researchers in the following ways. In those prior papers [26, 32], the original classification technique used for tracing architectural tactics in the source code was proposed; in this thesis, the training sets are the primary focus, in that the creation of these datasets presents a bottleneck problem in this area of research. In response to that bottleneck, novel Web-Mining and Big-Data Analysis methods are presented here for the aid of software traceability researchers in creating training sets for their uses. A series of empirical experiments are also included for evaluating the accuracy of these automated dataset generation techniques.

1.4 Organization of the Thesis

The reminder of this Thesis is organized as following: Chapter 2 provides the background for the architecture traceability problem and the related work in the area of automated dataset creation. In chapter 3 and 4 we present the overview of the three training set creation techniques and the design of the empirical experiments that were used. Then chapter 5 present the results of our comparison study. We also study the impact of training-set size on accuracy of creating trace links in chapter 6. Chapter 7 provides additional usage scenarios where the proposed automated techniques can be used, while Chapter 8 discusses threats to validity. Finally, we present our conclusions and future work in Chapter 9.

Chapter 2

Background and Related Work

This chapter reviews the thesis primary area of focus, software architecture traceability, and summarizes the related work in the areas of automated dataset creation and dataset augmentation.

2.1 Software Architectural Tactics

Software architectural tactics are one of the basic components of software architecture. Used to satisfy specific requirements and qualities for the software, a formal definition of these tactics is provided by Bachman et al. [6] who define a tactic as a means of satisfying a quality-attribute-response measure by manipulating some aspects of a quality attribute model through architectural design decisions. In essence, they are a specific means of satisfying a particular need of the software by structuring the software in a particular way to fulfill that requirement.

The focus of this research is limited to five specific tactics, in order to keep the scope of this research manageable, and were selected based on representing an array of reliability, performance, and security requirements. These specific tactics are: *heartbeat*, *scheduling*, *resource pooling*, *authentication*, and *audit trail*.

They are defined as follows [7]:

- **Heartbeat**: A reliability tactic for fault detection, in which one component (sender) emits a periodic heartbeat message while another component listens for the message (receiver). The original component is assumed to have failed when the sender stops sending heartbeat messages. In this situation, a fault correction component is notified.

- **Scheduling**: Resource contentions are managed through scheduling policies such as FIFO (First in first out), fixed-priority, and dynamic priority scheduling.

- **Resource pooling**: Limited resources are shared between clients that do not need exclusive and continual access to a resource. Pooling is typically used for sharing threads, database connections, sockets, and other such resources. This tactic is used to achieve performance goals.

- **Authentication**: Ensures that a user or a remote system is who it claims to be. Authentication is often achieved through passwords, digital certificates, or biometric scans.

- **Audit trail**: A copy of each transaction and associated identifying information is maintained. This audit information can be used to recreate the actions of an attacker, and to support functions such as system recovery and non-repudiation.

2.2 Traceability Challenge: Identifying Tactic-Related Classes

In this section, the classification-based method for tracing software architectural tactics in the source code is presented. A significant portion of the remainder of this thesis involves the training and output of this classifier, and analysis of the accuracy thereof. This training is accomplished using training datasets generated by various dataset creation methods.

In prior work [27, 32], a novel approach was presented by M. Mirakhorli et.al. for

tracing architecturally significant tactics, implemented in well-tested and therefore common software structures. As a tactic is not dependent upon a specific structural format, we cannot rely on structural analysis to identify them. Instead, the approach used relies on information retrieval (IR) and machine learning techniques to teach a classifier to recognize specific terms that occur commonly in relation to implemented tactics. By using this method, all classes related to a given tactic can be identified and then traced to the specific quality requirements and design rationales that drove the implementation of the given tactic [29].

Three phases of preparation, training and classification are part of the tactic classifier, defined as follows:

Data Preparation Phase: Data is pre-processed, using standard IR processing retrieval methods, such as stemming, removal of stop terms and so forth, and are represented as a vector of terms.

Training Phase: This phase involves feeding the classifier a set of preclassified code segments as input, with an output of indicator terms that are considered to be representative of and associated with each given tactic type. For example, the *scheduling* tactic will be more commonly associated with specific terms, such as priority, than those terms will be associated with other tactics. As such, the term *priority* can therefore be given a higher weighting in association with that tactic.

More formally, let q be a specific tactic such as *heart beat*. Indicator terms of type q are mined by considering the set S_q of all classes that are related to tactic q. The cardinality of S_q is defined as N_q. Each term t is assigned a weight score $Pr_q(t)$ that corresponds to the probability that a particular term t identifies a class associated with tactic q. The frequency $freq(c_q, t)$ of term t in a class description c related with tactic q, is computed for each tactic description in S_q. $Pr_q(t)$ is then computed as:

$$Pr_q(t) = \frac{1}{N_q} \sum_{c_q \in S_q} \frac{freq(c_q, t)}{|c_q|} * \frac{N_q(t)}{N(t)} * \frac{NP_q(t)}{NP_q} \tag{2.1}$$

This equation 2.1, Normalizes the frequency with which term t occurs in the training document with respect to its length. Also, it computes the percentage of training document

of type Q containing term t. The last fraction of this equation decreases the weight of terms that are project specific.

Classification Phase: This phase involves taking the indicator terms computed in the prior phase (Equation 2.1), and using them to calculate and evaluate the likelihood ($Pr_q(c)$) that a given class c is associated with the tactic q. Let I_q be the set of indicator terms associated with tactic q that were identified during the training phase. The classification score that class c is associated with tactic q is then defined as follows:

$$Pr_q(c) = \frac{\sum_{t \in c \cap I_q} Pr_q(t)}{\sum_{t \in I_q} Pr_q(t)} \tag{2.2}$$

The numerator is computed here as the sum of the term weights of all type q indicator terms that are contained in c, and the denominator is the sum of the term weights for all type q indicator terms. The probabilistic classifier for a given type q will assign a higher score $Pr_q(c)$ to class c that contains several strong indicator terms for q.

Tactic classifiers that rely on Information Retrieval (IR) provides a solution to the traceability challenge of connecting a tactic to its implementation in the source code. This technique requires a rich set of training data.

2.3 Related Work on Dataset Creation/Augmentation

The research area of data mining and information retrieval is presently very active, with many works of research into facilitating training set *selection* in text classification problems. These works of research specifically focus on the assumption that a large number of labeled data points exist, and from these data points, these works attempt to integrate various methods, including sampling [35, 37], instance selection [9, 11, 18], and data reduction methods [33, 37] to obtain a small, statistically representative sample of the larger data set. Unlike these approaches, we do not make such assumption and the main problem in the area of software traceability is the lack of any labeled data.

The area of requirements traceability has previously used web-mining in research. An

example of this use was by Jane Cleland-Huang et.al. [20], who used web-mining for sourcing an augmented query from the web to replace a difficult-to-retrieve query. That work did not focus on the creation of datasets, but was instead focus on expanding the terms within a requirement or requirements into more domain-specific terms, allowing for easier detection by traceability techniques. The methods in this thesis, in contrast to the prior method, is to create a code-based training set for architectural tactics. Anas Mahmoud [24] similarly used query augmentation techniques based on text clustering for the purposes of classifying requirements that do not have code functionality. This method uses semantically similar terms extracted from sources such as Wikipedia for augmentation of the sorting and classification of the text for the non-functional requirements.

In the context of data mining, Zarei et al [38] described the automatic generation of training datasets for classifying Peer-to-Peer (P2P) traffic. The generation of training samples was made through sampling packets from an incoming traffic and selecting some of them based on heuristics and statistical models. One advantage of this approach is that it could continuously update the classifier through collecting data periodically, so allowing their classification mechanism to detect new traffic patterns. By applying their dataset to classify incoming packets to a university network, their classifier was able to detect the traffic flow with higher than 98% accuracy when using the generated training data with a false positive rate of about 1.3%. These generated datasets, however, can be applied only for P2P traffic classification while our approach aims to solve the problem of reusable datasets for the software architecture domain.

Other research studies proposed techniques for automatically generating training-samples to improve their classification accuracy by increasing the size of the training datasets. For instance, Varga and Bunke [10] showed a training set generation for text recognition of handwritten documents. Their training data was generated through performing geometrical transformations into existing samples of handwritten lines of text. Through a set of experiments, they showed that the use of the generated training data increased the recognition

rate of handwritten text. Similarly, Guo and Viktor [21] approached the problem of adjusting unbalanced training data (when the number of samples from one class is significantly higher than the others) through the automatic generation of training samples from existing ones in the form of short-sized class of datasets. Contrasted to our approach, these solutions still need the manual collection of labeled training samples in order to produce more training data sets whereas our method does not need such samples. In fact, they were concerned about increasing training data size rather than providing a solution to automatically generate training data samples for being reused by classifiers within the same domain.

A number of projects have recently (as of 2015) been examining the area of data-mining ultra-large-scale open source software repositories [17, 39], with the intent of focusing on studying source code and coding issues across large numbers of projects. There are a limited experimental research projects on using such ultra-large-scale code repositories for the generation of scientific datasets, and an even smaller number of those focus on research into software requirements and architecture research. Extensive searching in the published literature has led the researchers to conclude that there are presently no concrete techniques for automated dataset generation for scientific research in this area.

While automated dataset generation is apparently a neglected area of research, there are a number of independent software engineering communities that are providing methods and mechanisms for sharing and publishing what datasets are generated in the course of research. The Mining Software Repositories (MSR) conference holds an annual Data Track for researchers to publish and share their datasets, while the Center of Excellence for Software Traceability holds regular traceability challenges; at these challenges the researchers can share their software traceability datasets. However, most of the datasets found in these repositories were manually created by experts [23]. While useful, this thesis is focused on taking massive quantities of publically available data from the Internet and from ultra-large-scale software repositories and using automation for the creation of high-quality datasets.

Chapter 3

Three Methods for Training-Data Creation

As mentioned in chapter 1, this thesis presents novel automated techniques for the purpose of creation of training sets for algorithmic analysis and tracing of architectural tactics. These automated techniques are designed so that the outputted data-sets can be created with minimal compiling costs, especially in comparison with the high labor costs associated with datasets created manually by expert software developers, and are aimed to have comparable quality with such expert-sourced datasets.

The two novel proposed methods for training set creation, as well as the industry standard expert-based method, are illustrated in Figure 3.1. A summary of the three methods is as follows: in the Manual Expert-Based approach, developers or architects collect, review and refine the training set directly. For the two automated techniques, a description of the specific tactic, which can be sourced from relevant textbooks (or a set of tactic-related terms) can be used as a search query in the relevant database. Then, for both approaches, advanced searching and filtering techniques are used to identify API descriptions or actual implementations of the tactic from the technical libraries or open source software repositories, as appropriate to the given method.

The following sections describe each of these three techniques in greater detail, with Chapter 4 consisting of reports on empirical studies conducted to compare them.

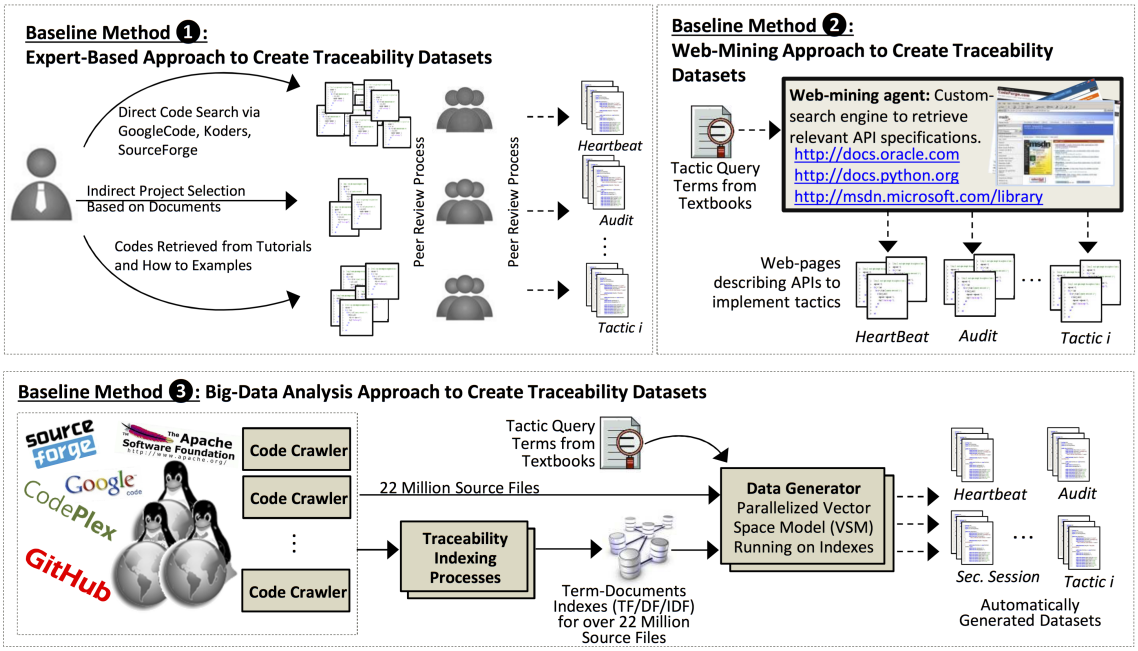


Figure 3.1: Overview of Automated Approaches to Create Tactic Traceability Training-sets

3.1 Method 1: Expert-Centric Approach

In previous work [31, 32] by M. Mirakhorli et.al., a manual approach was used to collect datasets for the purpose of training the tactic classifier algorithm. The training set shown in Table 3.1 was compiled by software architecture and requirements engineering experts, and then peer reviewed by an additional two independent evaluators, with each of the experts involved possessing between two to eight years of experience in the area of software architecture.

The code snippets that comprised the dataset, each snippet being an implementation of an architectural tactic, were compiled via the following process:

- *Direct Code Search :* The source code search engine *Koders* [2] was utilized, with the search query being comprised of keywords used in descriptions of the tactic. These descriptions were sourced from academic textbooks, industry articles, white papers, and code libraries belonging to the architects from previous implementations of the tactic in question. Once the search results were returned, each of the code snippets

Table 3.1: Manual Dataset Generated by Expert

Tactic	**Projects**
Audit	1-Jfolder(Programming), 2-Gnats(Bugs Tracking), 3-Java ObjectBase Manager(Database), 4-Enhydra Shark(Business, workflow engine), 5-Openfire aka Wildfire(Instant messaging), 6-Mifos(Financial), 7-Distributed 3D Secure MPI(Security), 8-OpenVA.(Security), 9-CCNetConfig(Programming), 10-OAJ (OpenAccountingJ)(ERP)
Scheduling	1-CAJO library(Programming), 2-JAVA DynEval(Programming), 3-WEKA Remote Engine(Machine Learning), 4-Realtime Transport Protocol(Programming), 5-LinuxKernel(Operating Systems), 6-Apache Hadoop(Parallel Computing), 7-ReactOS(Operating Systems), 8-Java Scheduler Event Engine(Programming), 9-XORP(Internet Protocol), 10-Mobicents(Mobile Programming)
Authentication	1-Alfresco(Content management), 2-JessieA Free Implementation of the JSSE(Security), 3-PGRADE Grid Portal(Business, workflow engine), 4-Esfinge Framework(Programming), 5-Classpath Extensions(Workflows Management), 6-Jwork(Programming), 7-GVC.SiteMaker(Programming), 8-WebMailJava(Programming), 9-Open Knowledge Initiative(OKI)(Education), 10-Aglet Software Development Kit(Programming)
Heartbeat	1- Real Time Messaging-Java(Programming), 2-Chat3(Instant messaging), 3-Amalgam(Content Management), 4-Jmmp(Programming), 5-RMI Connector Server(Web Programming), 6-SmartFrog(Parallel Computing), 7-F2(Financial), 8-Chromium NetworkManager(Web Programming), 9-Robot Walk Control Behavior(Programming), 10-Apache(Programming)
Pooling	1-ThreadPool Class(Programming), 2-Open Web Single Sign On(Web Programming), 3-ThreadStateMapping2(Programming), 4-RIFE(Web Programming), 5-Mobicents(Mobile Programming), 6-Java Thread Pooling Framework(Programming), 7-Concurrent Query(Programming), 8-RIFE(Web Programming), 9-RIFE(Web Programming), 10-EJBs(Web Programming)

contained therein were reviewed by two other experts to determine if the code was relevant to the present tactic or not.

- *Indirect Code Search :* This search method involved combing various project-related documents (design documents, online forums, etc.) for references and pointers to the specific architectural tactics. In this search, experts checked the project's website and searched within their architectural documents looking for any implemented tactics. Once they find that a tactic is implemented in the project, they conduct a code review to see which part of the code is implementing this tactic. After finding the code-snippet of the tactic within the source code, they save it and add it to the training-set. Results from these searches were then used to identify and collect relevant code snippets, which were then peer reviewed by a quorum of experts for confirmation of relevance to the specific tactic.

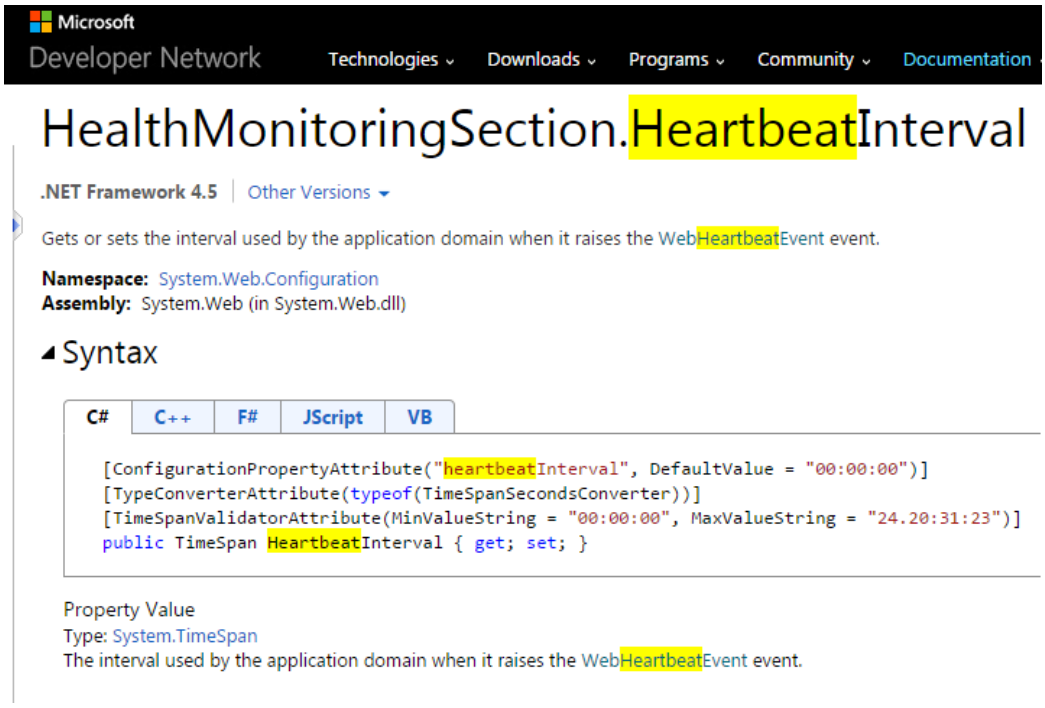
- *"How to" examples :* This search method involved examining various online materials, online libraries (e.g. MSDN), technical forums (such as Stack Overflow) and tutorials for specific examples of implementations of the selected architectural tactics.

While the rigorous search and validation approach used in this manual data collection resulted in a high quality and precise traceability dataset, it also had significant costs in time and effort. The project researchers required 3 months of collection and subsequent peer review of the code snippets from 10 different projects for the 5 architectural tactics selected.

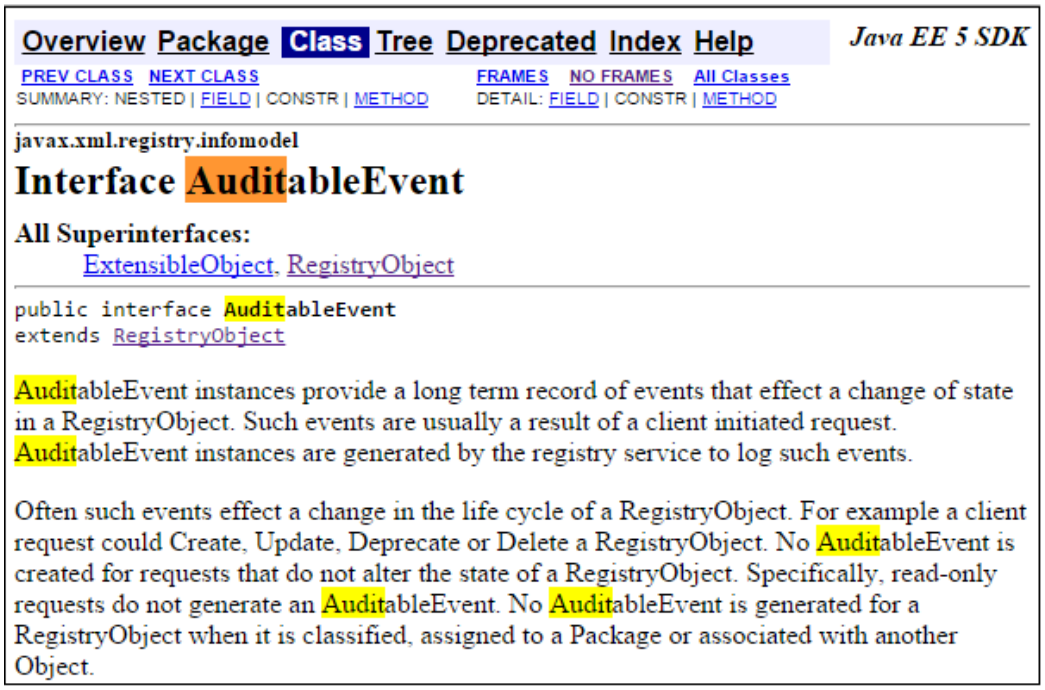
3.2 Method 2: Web-Mining Approach

Online-based coding libraries, such as *msdn*[1] or *oracle*[2], are potential resources for information regarding the implementation of architectural tactics, in addition to other areas

[1] https://msdn.microsoft.com
[2] http://www.oracle.com



(a) implementing reliability concerns through Heartbeat tactic from msdn.com



(b) addressing security concerns through Audit Trail tactic from Oracle.com

Figure 3.2: Two sample API descriptions from technical libraries of (a) MSDN and (b) Oracle

of design and programming considerations.

The initial hypothesis was that high-quality classifier training sets would result from processing these libraries. Figures 3.2(a) and 3.2(b) diagram representative implementation diagrams that were sourced from these libraries for the implementation of reliability requirements via the *Heartbeat* tactic and security requirements through *Audit Trail* tactic.

3.2.1 Data-Collection Agent

A custom web-scraper was developed, using the Google search engine APIs for querying predefined online technical libraries (e.g. *msdn* and *oracle*) for their related content.

For this method, the search queries were designed around keywords relevant to the tactic in question, with the keywords sourced from textbook descriptions of the relevant tactic. Regarding the *HeartBeat* tactic, for example, the following textbook description was used [7]: "***Heartbeat*** *is a* ***fault detection*** *mechanism that employs a periodic message exchange between a system* ***monitor*** *and a process being monitored.*" The trace user generated the following *trace query* from this description: *Heartbeat* OR *fault* OR *detection* OR *monitoring*.

These searches collected a number of high-relevancy web-pages in regards for each of the specific tactics, with the scraper agent returning ranked web-pages that contained both API documentation and sample code snippets relevant to the tactics implementation. A basic degree of pre-processing, in the form of a simple HTML tag filter, is part of this process, so that the HTML tags are removed and only the textual content remains, which is then stored in a plain text file.

3.2.2 Data Generation

Using the information gathered by the web-scraper searches, the dataset is compiled. These classifier training datasets contain the pre-processed web page contents in the form of balanced sample text files that are either tactic-related (positive samples) or non-tactical (negative samples). Although the Web-Mining approach is able to generate unbalanced

training sets, for the sake of comparing different baseline techniques we generate balanced datasets. As in the automated approached we are generating the same number of positive and negative samples as the manual approach.

API documentations regarding tactics and sample tactical code snippets compose the positive samples, while the negative/non-tactical samples are sets of highly-dissimilar documents in relation to the original query. These negative samples aid in reduction and removal of terms that are over-represented in the library web-pages (e.g. Microsoft in the MSDN library).

3.3 Method 3: Big-Data Analysis Approach

In this approach, for the creation of classifier training sets, ultra-large-scale open-source repositories were data-minded using machine-learning methods. An overview of this approach and its various components are illustrated in Figure 3.1.

3.3.1 Creating Open Source Projects Repository

The initial component of this approach is the source-code scraper, which is used for the actual data-mining of the various source-codes from the open-source software repositories.

The ultra-large-scale open source repository used for this study was compiled by extracting 116,609 unique projects from various open-source software repositories around the Internet–Github, Google Code, SourceForge, Apache and other such websites, using a variety of code-crawler applications that we have developed for this extraction. Githubs projects, for example, were extracted by means of a torrent system, known as GHTorrent[3], which acts as a data and event extraction service for the community, releasing the extracted information in the form of MongoDB data dumps. These dumps contain information about the Github projects in the form of various community updates–user updates, commit comments, languages, pull requests, follower-following relations and other such user-centric

[3]http://ghtorrent.org/

data.

The automated parsing/fingerprinting crawler application, *Sourcerer*, developed by University of California, Irvine, researchers [36], was also used. Projects from Apache, Java.net, Google Code and SourceForge were extracted using this application. The resulting repository contained various versioned source code, from multiple releases, relevant documentation (when and if available), project metadata, and a coarse-grained structural analysis for each project.

Having gathered all of these projects from these open source data repositories, the data was cleaned, removing all empty or very small projects. Table 3.2 shows the resulting frequency of projects in different languages contained in the resulting repository after this processing.

3.3.2 Data Indexing

Once the data has been compiled, the next stage in the Big-Data Analysis method is a term-document indexing component. This component indexes the occurrence of terms across the array of source files from the projects within the compiled code repository. This component, called *Traceability Indexing*, involves several stages of pre-processing. This processing entails removal of stop words, then taking the remaining terms and stemming them to the root form for each word, and then indexing the source files. The index then stores statistics regarding each document and the source files; these statistics include *term frequency (TF)*, *document frequency (DF)*, *TF/IDF (Inverse Document Frequency)* and *location of source file*. These statistics aid in making term-based searches more efficient, and allowed for the index to be used as an inverted index; any given term can be selected and shown which source files contain that term [25].

Table 3.2: Overview of the projects in Source Code Repository of Big-Data Analysis Approach

Language	Freq.	Language	Freq.	Language	Freq.	Language	Freq.	Language	Freq.
Java	32191	Go	1614	Emacs Lisp	321	ActionScript	120	F#	74
JavaScript	22321	CoffeeScript	1187	Visual Basic	134	Elixir	82	Kotlin	43
Python	9960	Scala	729	Erlang	154	Scheme	80	Bison	39
CSS	9121	Perl	699	Processing	152	Prolog	77	Cuda	37
Ruby	8723	Arduino	321	PowerShell	151	D	72	LiveScript	32
PHP	8425	Lua	458	TypeScript	139	Common Lisp	65	AGS Script	29
C++	5271	Clojure	456	OCaml	105	Pascal	60	SQF	26
C	4592	Rust	308	XSLT	102	Haxe	60	Mathematica	25
C#	4230	Puppet	286	ASP	85	FORTRAN	45	Apex	22
Objective-C++	33	Groovy	253	Dart	84	OpenSCAD	44	PureScript	22
Objective-C	2616	SuperCollider	185	Julia	84	Racket	44	DM	21

*Total number of projects:116,609, *Total number of source files: 23M

3.3.3 Data Generator Component

The third stage-component of the Big-Data Analysis method is a paralleled version of a Vector Space Model (VSM) [34] . This VSM is capable of processing many millions of source files in a matter of a few seconds, and is a standard method for comparing vectors of weighted terms and computing the cosine similarities between a query and a document. More complete information and explanations regarding VSM systems can be found in a typical textbook on information retrieval programming [34].

For the purposes of this study, the VSM is used to create a dataset related to the software tactic, based on a query instigated by a trace user. For example, the query ”Secure Session User Access Session Management” was used to search for code-snippets of Secure Session tactic. The cosine similarity score calculated is directed between that provided query and the set of the combined source files contained within the ultra-large-scale software repository, and then for each tactic, the most relevant source files exhibiting highest similarity to the trace query are selected. To prevent the addition of domain-specific files, an n number of samples (n being user-defined) of non-tactical files for each tactic from the same projects as the high-similarity cosine files are also added. Previous experience and testing has demonstrated that the inclusion of unrelated sample data has a statistically significant effect on the quality of the classifier training for indicator terms [15, 27, 32].

3.4 Generated Data-Set

The dataset generated from this data contains a balance between tactic-related and unrelated code snippets, retrieved from 10 open-source projects, with each project contributing one tactic-related file and one non-tactic-related file. As for the sake of comparing the three approaches we used a balanced datasets.

Chapter 4

Experiment Design

In this chapter, the experimental design for comparing the performance of the three baseline training-set creation methods is presented. Concurrently with this presentation, the research questions discussed earlier will be answered. The justifications behind the selection of these techniques will also be discussed, along with the details of the methodology used in the course of this comparison and the validation of the results of the comparison.

4.1 Approaches Selection

Research into automated generation of training sets is new in the area of computer science. While there have been prior studies into the areas of trace-query replacement and augmentation, so far as research has indicated, the concept of automated generation of such training sets has not been explored in the published literature.

Research and development into this area relies on the creation and maintenance of large-scale, (un)structured and data-rich knowledge bases; these criteria are found in both online databases and ultra-large-scale source code repositories, this thesis has proposed the novel approach of using these resources in the development of this concept. With the resulting new techniques, it is hoped that development in this area of research will result in the consistent automated creation of high-quality code datasets.

4.2 Oracle Dataset Used as Testing set

The manually compiled expert-created dataset of architectural tactics was used as the baseline testing set; the other methods would be judged in comparison against the results from the expert-set. As a note on this dataset, the compiling phase, during which the various code snippets were manually collected and peer-reviewed by software experts, lasted for three months.

A total of 50 open source projects were found by the experts, ten for each tactic of interest, in which the specific tactic was implemented. Each project was then subjected to a software architecture biopsy, with a manually-extracted source file containing the implementation of the specific tactic being extracted, along with one randomly chosen non-tactical file. As a result, the training set, when fully compiled, consisted of 100 total files, 50 tactic-related snippets (10 for each tactic), and 50 non-tactic-related snippets.

4.3 Experiment Design

Three experimental designs were created in order to answer the previously stated research questions regarding the baseline-technique comparisons.

4.3.1 Experiment Design for Expert-Based Method

The Expert-based method and the accuracy of the classification techniques trained using this method were evaluated using a standard 10-fold cross-validation process. The expert-compiled code-snippet dataset was utilized as the training set and the testing set. This form of evaluation is considered to be a classic, and is extensively used in the areas of data-mining, information retrieval, and automated requirements traceability [13, 14, 22, 32].

For each run of the experiment, the dataset was partitioned by the originating project, such that the first nine of the ten projects (each including one related and four unrelated code-snippets) were used as the training set, with the final tenth project being used as a test of the training. This was done ten times in total, with each project being used in the

training nine times and being used as the test classifier once. The experiment was then repeated, using the same pairs of classification thresholds and term thresholds as in the prior experiment.

4.3.2 Experiment Design for Web-Mining Method

The design of the experiment for the Web-Mining method was to use the web-mining technique to extract code-snippets from online technical libraries, including*MSDN* and *ORACLE*. Using this dataset, the tactic classifier was trained, and then tested against the expert-compiled code-snippet dataset (table 3.1). This experiment was repeated to allow for the testing of a variety of term thresholds and classification thresholds required for Formula 2.1 and Formula 2.2.

4.3.3 Experiment Design for Big-Data Analysis Method

The final baseline experiment, using the Big-Data Analysis method, involved the classifier being trained on a dataset compiled from the ultra-large-scale source code repository (containing over 116,000 open source projects). Once trained on this dataset, the classifier was then tested against the manually-compiled expert-created code snippet dataset.

4.4 Evaluation Metrics

The results from the experiments were evaluated using four standard metrics for recall, precision, f-measure, and specificity. These were calculated according to the following formula (*code* in these formula is a contraction of the term *code snippets*)

$$Recall = \frac{|RelevantCode \cap RetrievedCode|}{|RelevantCode|} \tag{4.1}$$

while precision measures the fraction of retrieved code snippets that are relevant and is computed as:

$$Precision = \frac{|RelevantCode \cap RetrievedCode|}{|RetrievedCode|} \tag{4.2}$$

As it is not practical nor feasible to expect identical recall values between successive iterations of the algorithm, the f-measure calculates the the harmonic mean of recall and precision and can be used to compare results across experiments:

$$FMeasure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4.3}$$

Lastly, specificity measures the fraction of unrelated and unclassified code snippets. It is computed as:

$$Specificity = \frac{|NonRelevantCode|}{|TrueNegatives| + |FalsePositives|} \tag{4.4}$$

4.5 Minimizing Biases

To mitigate the impact of the datasets size, all of the automatically generated datasets used in the above experiments were identical in size to the expert-generated dataset, including 10 projects (or 10 related web-pages, where appropriate). We trained the classifier

using the code snippets automatically extracted using our own primitive big-data analysis technique and then attempted to classify the accurate dataset of manually established and reviewed code snippets.

Minimizing the variables and preventing a bias regarding dataset size required that the automatically compiled datasets be set to equal in size to the manually-compiled training dataset. Therefore, each of these datasets only included 10 sample API specifications, with the addition of 40 descriptions of non-tactic-relevant IT documentation (collected by web-scraper) for training purposes.

Furthermore, to minimize the risk of biases towards the selection of tactic-related terms in the search query, terms from academic textbooks describing the tactics were gathered and added.

Other methods for reducing risks and threats to this research validity were approached in a systematic manner, and are discussed at length in chapter 8.

Chapter 5

Empirical-Study Results

The experiments described in chapter 4 were executed, training the tactics classifier using the three baseline approaches. The results were then compiled, with Table 5.1 reporting the top ten indicator terms that were learned for each of the five tactics for each of the three techniques. While there is significant duplication between these reports, it is unsurprising that the code-snippet methods learned more code-oriented terms such as *ping*, *isonlin*, and *pwriter*

The f-measure reports are shown in Figure 5.1, indicating the overall results for the classifier by tactic, using multiple combinations of the threshold value. In general, the three baseline methods all returned similar degrees of accuracy, with the Expert-compiled dataset outperforming the automated compiled datasets in two cases (*audit* and *heartbeat*). Conversely, for the authentication tactic, the Web-Mining-trained classifier achieved the same accuracy level as the manually compiled dataset classifier.

Regarding the *pooling* and *scheduling* tactics, the Big-Data-trained classifier was more accurate than either of the other two methods at term threshold values of 0.01 and 0.001 and classification thresholds of 0.7 to 0.3.

For clarification in these graphs, the horizontal lines in which there are no variations in f-measure score across various classification values. This tends to occur when all the terms scoring over the term threshold value also score over the classification threshold.

Table 5.1: Indicator terms learned during training

Tactic Name	**Web-Mining trained indicator terms**	**Big-Data trained indicator terms**	**Code trained indicator terms**
Heartbeat	nlb, cluster, balanc, wlb, ip, unicast, network, subnet, heartbeat, host	counter,fd, hb, heartbeat, member, mbr, suspect, ping, hdr, shun	heartbeat, ping, beat, heart, hb, outbound, puls, hsr, period, isonlin
Scheduling	schedul,parallel,task, queue,partition,thread, unord, ppl, concurr, unobserv	schedul,prioriti,task, feasibl, prio, norm, consid, paramet, polici, thread	schedul, task, prioriti, prcb, sched, thread, , rtp, weight, tsi
Authentication	authent,password, user,account,credenti, login, membership, access, server, sql	password,login, usernam,rememb, form, authent, persist,sign, panel, succeed	authent,credenti, challeng,kerbero, auth,login,otp, cred, share,sasl
Pooling	thread, wait, pool, applic, perform, server, net, object, memori, worker	pool, job, thread, connect, idl, anonym, async, context, suspend, ms	pool, thread, connect, sparrow, nbp, processor, worker, timewait, jdbc, ti
Audit Trail	audit, transact, log, sql, server, secur, net, applic, databas, manag	trail, audit, categori, observ, udit, outcom, ix, bject, acso, lesser	audit, trail, wizard, pwriter, lthread, log, string, categori, pstmt, pmr

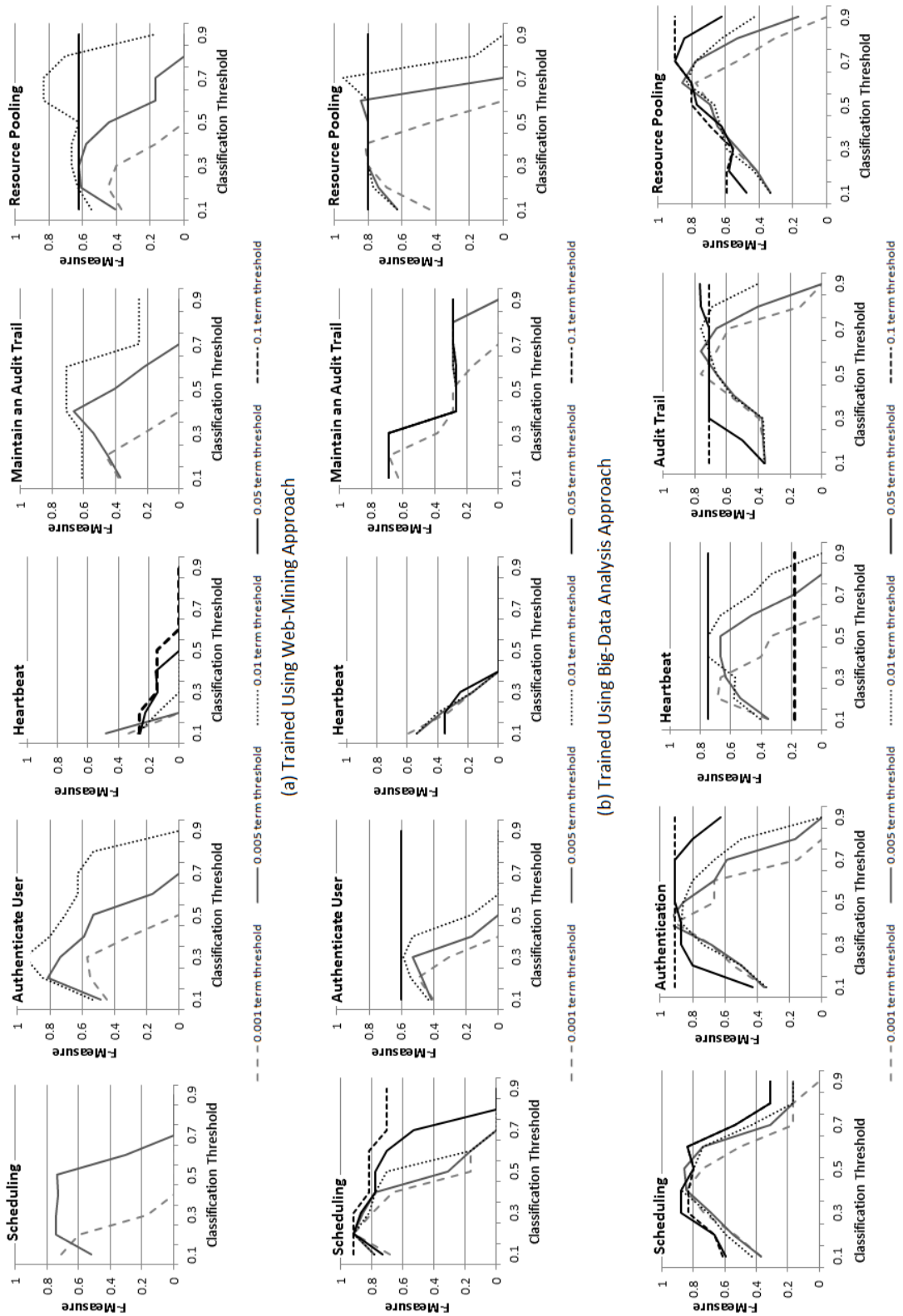


Figure 5.1: Results for Detection of Tactic-related Classes at various Classification and Term Thresholds for five Different Tactics

Table 5.2 reviews the optimal results for each of the tactics–that being, which of the results achieved a high level of recall (0.9 or higher, if practically possible) in conjunction with as high a precision result as possible. These results show that, in four cases, the classifier trained with the Expert-collected data succeeded at recalling the entire tactic-related classes, while also achieving reasonable precision.

Table 5.2: A Summary of the Highest Scoring Results

Tactic Name	**Training Method**	**FMeasure**	**Recall**	**Prec.**	**Spec.**	**Term/ Classification threshold**
Audit	Web-Mining	0.71	1	0.55	0.785	0.01 / 0.4
	Big-Data	0.687	1	0.523	0.762	0.001 / 0.2
	Expert-Manual	0.758	1	0.611	0.833	0.001 / 0.5
Authentication	Web-Mining	0.956	1	0.916	0.9772	0.01 / 0.3
	Big-Data	0.6	0.545	0.666	0.931	0.05 /0.1
	Expert-Manual	0.956	1	0.916	0.977	0.005 / 0.4
Heartbeat	Web-Mining	0.48	0.545	0.428	0.813	0.005 / 0.1
	Big-Data	0.592	0.727	0.5	0.813	0.001 / 0.1
	Expert-Manual	0.689	1	0.526	0.775	0.001 / 0.2
Pooling	Web-Mining	0.833	0.909	0.769	0.931	0.01 / 0.6
	Big-Data	0.952	.909	1	1	0.01 /0.7
	Expert-Manual	0.9	0.818	1	1	0.05 / 0.7
Scheduling	Web-Mining	0.740	0.909	0.625	0.863	0.005 /0.2
	Big-Data	0.916	1	0.846	0.954	0.001 / 0.2
	Expert-Manual	0.88	1	0.785	0.931	0.01 / 0.4

The Big-Data-trained classifier achieved recall of 0.909 in one case, and a recall of 1.0 for two of the tactics. The classifier trained using the Data-mining approach achieved a recall of 1.0 in two cases and 0.909 for two other tactics.

5.1 RQ 1: Manual Expert-Based Training vs. Automated Web-Mining.

These results indicate that, in 4 out of 5 cases, the manual Expert-Based approach outperformed the Web-Mining technique, albeit by a small margin. Table5.3 shows these differences between the f-measures for Expert-Manual and Web-Mining.

Based on this observation, while admittedly limited, it seems possible to compare the Expert-Based method as being equivalent to the Automated Web-Mining approach. In

Table 5.3: Differences in F-Measure of Expert-Based and Automated-Web Mining.

Audit	Authenticate	Heartbeat	Pooling	Scheduling
0.048	0	0.209	0.067	0.14

order to evaluate whether the differences were statistically significant, both *Wilcoxon* and *Friedman ANOVA* tests were performed; the latter of these is a non-parametric test for comparing the medians of paired samples (note that the data was not normally distributed). Both tests are recommended for analysis of small datasets (as small as 5 per group) [16].

In both statistical tests, the null hypothesis could not be rejected (there is a difference in median/mean-rank of two groups.) [1]

Result 1: There is no statistically significant difference between the trace link classification accuracy for a classifier trained using Expert-Based approach and Automated-Web Mining.

5.2 RQ2: Manual Expert-Based Training vs. Automated Big-Data Analysis.

For the classification of two out of the five tactics, the Big-Data Analysis method outperformed the manual Expert-Based approach. In two of the remaining cases, both methods returned extremely close results, with Table 5.4 showing the differences between the f-measure of the two approaches.

Table 5.4: Differences in F-Measure of manual Expert-Based and automated Big-Data Analysis Approach

Audit	Authenticate	Heartbeat	Pooling	Scheduling
0.071	0.356	0.097	-0.052	-0.036

[1] p-value of 0.05

Wilcoxon and *Friedman ANOVA* tests were again used to conduct a comparison between the medians of the paired samples. In both cases, the null hypothesis was retained.[2]

Result 2: There is no statistically significant difference between the trace link classification accuracy for a classifier trained using Expert-Based approach and automated Big-Data Analysis. This indicates that Big-Data Analysis approach can be used as a practical technique to help software traceability researchers generate datasets.

[2] p-value of 0.05

Chapter 6

Cost-Benefit Analysis

The empirical study of the three baseline methods for training-dataset creation suggests that there are no statistically significant differences in trace link accuracy between classifier algorithms trained using these methods. However, while the results seem to indicate no difference in the results, the resources required are very different, with the Manual Expert method requiring an impressive amount of highly trained labor.

Cost-Comparison In a prior work, the estimated time requirements for the creation of the training set using the Expert-Based approach for 5 tactics was 1080 hours. Taking into account the hourly salary of an expert, or even a student, in terms of dollars-per-hour will make this approach cost many tens of thousands of dollars.
In contrast, the automated Big-Data Analysis method generated similar code snippet datasets within a few seconds (with the primary time cost there being in the creation and maintenance of the ultra-large-scale code repository).

One drawback for any automated data-mining algorithm method is the inherent inaccuracy of these techniques. To detect if this inaccuracy affected this research, the automatically generated training datasets were manually evaluated after being compiled, with the accuracy, as determined by this evaluation, being shown in Table 6.1.

Overall, the Big-Data Analysis method was more accurate than the Web-Mining method in the number of correct data-points, in the form of accurate code-snippets. A possible explanation for this discrepancy might be from a higher signal-to-noise ratio in the online technical libraries, or it might be because of inaccuracies in the basic search technique used in the Web-Mining approach. Determining the precise cause is a potential area of focus for future work.

Table 6.1: Accuracy of automatically generated training-set

	Audit	Scheduling	Authentication	Heartbeat	Pooling
Web-Mining	0.6	1	0.91	0.6	0.8
Big-Data Analysis	1	1	1	0.9	0.9

The data quality between Big-Data Analysis and the Manual Expert method were also compared. The Big-Data method successfully retrieved relevant code snippets from the large scale software repository in over 90% of the cases. In contrast, the manually collected training-sets was of universally higher quality, as it not only contained 100% accurate data points (due the rigorous data collection), the experts had also taken into account the representativeness, diversity, and generalizability of these segments of code, as well as the quality of the collected samples for the intended training purposes. The manually collected code snippets were also richer in terms of contained vocabularies, APIs and comments. Based on this observation, it is believed that this is one of the primary differences between the baseline methods.

Further investigating the scores assigned to each indicator term across the three baseline techniques, it was observed that the indicator terms generated by the manually created dataset possessed greater probability scores, and were organized in a more coherent, less noisy (e.g. fewer unrelated terms) fashion. In future work, it is intended to augment the automated methods so that these methods will not only find related code-snippets, they will also take into account other metrics related to data quality and sampling strategies.

6.1 To What Extent Expert-Based Approach Can be Practical?

In the research community, there is an ongoing debate regarding research techniques developed, based on, or examined using student-generated datasets. One aspect of this debate is the need for using various techniques for mitigation of the datasets inherent biases and research threats resulting from these methods [20, 23]. On the positive end, however, the community is also invested in the concept of the Expert-compiled dataset, of which the student-generated datasets are related. In this section, the problems and challenges in this area will be examined. **Hypothesis :** A general assumption in the area of machine learning is the following statement: *the larger the size of training set, the more accurate and generalizable the underling learning method will be*.The basis of this belief is that, as the sample size grows, it will more accurately reflect the population it is sourced from, and therefore the sample will be distributed more closely around the populations mean.

However, in the case of machine learning datasets, this assumption has not yet been tested, meaning that it is unknown if there is a net benefit in extending the training set size when generated by the experts. As such an extension means that more resources will be required in order to generate the larger dataset, it can and will have a significant cost. In contrast, with the variety of mitigation techniques used to minimize the threat to validity and create generalizable training sets, this assumption is ripe for scientific testing. **Experiment to Investigate:** For this experiment, the effect of varying dataset sizes on the accuracy of traceability link discovery is tested. The intent of this experiment is not to disprove or prove the importance of training set size, but to allow for a cost-benefit curve to be plotted for optimum tradeoff between dataset size and results, which is of significant importance when the dataset must be manually compiled.

In this case, the extension of the training set from 10 open source projects to 50 projects required the addition of six additional months of searching and peer review. As such, investigating at what point diminishing returns are reached is of significant importance.

6.1.1 Experiment Design

In prior work in this area of research by M. Mirakhorli et.al. [32], training sets of code snippets from 10 software systems were used. In an extension of this work [27] code-snippets sampled through via a peer-reviewed process from 50 open source projects were used. These training sets were established using systematic manual peer review, resulting in an extensive amount of time and effort spent in their creation.

Dataset For each of the five tactics included in this study, three experts identified 50 open-source projects in which the tactic was implemented. For each project an architecture biopsy was performed to retrieve the source file in which each utilized tactic was implemented. In addition, for each project a randomly selected non-tactical source files was retrieved.

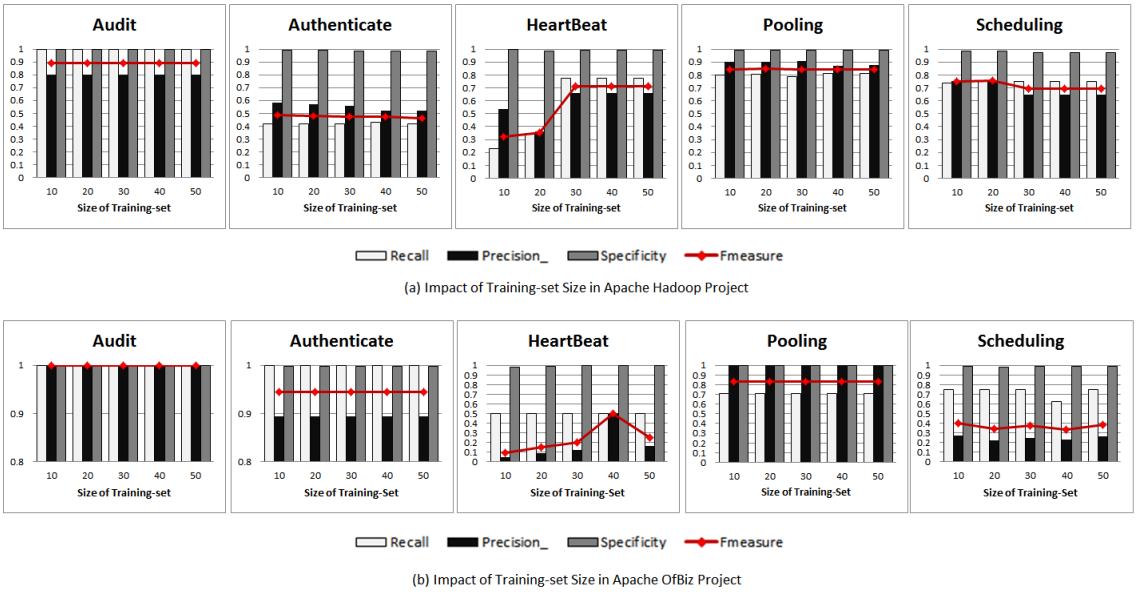


Figure 6.1: The impact of training-set size in manually established dataset on accuracy of recovering trace links

Impact of Training set Size on Trace Link Accuracy (Two Case Studies) In this section of research, RQ3 is investigated: What is the Impact of Training set Size on the Accuracy of Trace Link Classification?

On the first pass, the classifiers were trained using 5 sub-samples of the larger dataset,

allowing for the size ranges of 10, 20, 30, 40, and 50 sample code snippets to be tested. Each time, the classifiers were also used against the source code of Apache Hadoop and OfBiz. These two projects are used in industry and are considered to be representative of complex software systems.

Once complete, the trace-link accuracy of classifiers trained using different training set sizes was compared. This aided in investigating if there is an optimum return-on-investment for employing experts to establish large(er) training sets.

The accuracy metrics of this experiment are reported in Figure 6.1. The bars in this graph show precision, recall and specificity [32]. The red line indicates the f-measure metric. With the noted exception of the heartbeat architectural tactic, which exhibited major changes across the different training set sizes, the other four tactics did not show any significant changes in the accuracy of trained classifier.

Result 3: This observation supports the concept that, when manually creating a high quality training set, the size of dataset will not automatically have a significant impact on the accuracy of the classification technique described in Equations2.2 and 2.1. Therefore, collecting additional data-points by experts will not automatically increase the accuracy or generalizability of the trained classifier.

It should be noted that this observation is only supported by the data obtained from two case studies. Additional experiments to validate this result are a potential area of investigation for future work.

Chapter 7

Application to other Areas of Requirements Engineering

The prior experiments reported in this thesis show the potential in automated training set generation methods, and that they are feasible and potentially practical. The results of these experiments indicate that the Web-Mining method and Big-Data Analysis method can, in fact, automatically generate training datasets that are on par with manually-compiled expert-created datasets.

This chapter of this thesis reports on a feasibility study regarding the potential of using the proposed automated dataset creation techniques for supporting research into different areas of requirements engineering. This study will provide an answer for RQ4 (stated in chapter 1).

7.1 Usage Scenario#1: Tracing Regulatory Codes.

Here is presented the first potential usage scenario in the application of automated dataset generation techniques in the area of tracing regulatory codes. **Problem:** One of the complications faced by the research community in the subject area of requirements traceability is the lack of datasets on details for requirements, implementations and/or documentations as they relate to regulatory codes within a given software domain. There are a limited number of datasets commonly used in this area, such as Certification Commission

for Health care Information Technology (CCHIT) or the Health Insurance Portability and Accountability (HIPAA) (which can be found on the COEST.ORG website).

The proposed research techniques in this area have been primarily evaluated by running experiments over sections of the same dataset, or by tracing one of these through the source code of the two open source software systems of WorldVista and Itrust.

Feasibility Study: Quality technical libraries, such as MSDN, possess an array of documentations, technical guidelines, best practices, preselected technologies, and APIs. These can be used to address a wide range of regulatory codes, such as HIPAA and SOX [8]. For example, Table 7.1 lists a set of regulatory compliance acts which have had significant technical discussions regarding them in the MSDN library.

In a preliminary study, the automated technique presented previously was used to create a dataset for the domain of Tracing Regulatory Code. A sample experiment was run to create a dataset for technologies which could be used to address HIPAA regulations related to Database and Security. Once complete, the accuracy of the extracted data was evaluated; the results of this evaluation are presented in table 7.2, with 63% of the automatically generated data points being indicated as correct.

Due to space considerations, only two sample data points are excerpted here:

- *"HIPAA compliance: Healthcare customers and Independent Software Vendors (ISVs) might choose SQL Server in Azure Virtual Machines instead of Azure SQL Database because SQL Server in an Azure Virtual Machine is covered by HIPAA Business Associate Agreement (BAA). For information on compliance, see Azure Trust Center."*

- *"Confidentiality: Do not rely on custom or untrusted encryption routines. Use OS platform provided cryptographic APIs, because they have been thoroughly inspected and tested rigorously. Use an asymmetric algorithm such as RSA when it is not possible to safely share a secret between the party encrypting and the party decrypting the data...."*

Table 7.1: Sample Regulations Discussed on Technical Libraries

ACT Name	Aplies to
Sarbanes Oxley Act	Legislation passed by the U.S. Congress to protect shareholders and the general public from accounting errors and fraudulent practices in the enterprise, as well as improve the accuracy of corporate disclosures [8]. More on (http://www.sec.gov/)
HIPAA	the federal Health Insurance Portability and Accountability Act of 1996. The primary goal of the law is to make it easier for people to keep health insurance, protect the confidentiality and security of health care information and help the health care industry control administrative costs. [8]
PCI	Payment Card Industry Data Security Standard(PCI DSS) is a proprietary information security regulation for organizations that handle branded credit cards. [4]
The Gramm-Leach-Bliley Act (GLBA)	Also known as the Financial Services Modernization Act of 1999, is an act of the 106th United States Congress, removing barriers for confidentiality and integrity of personal financial information stored by financial institutions. [3]
SB 1386	California S.B. 1386 was a bill passed by the California legislature. The first of many U.S. and international security breach notification laws. Enactment of a requirement for notification to any resident of California whose unencrypted personal information was, or is reasonably believed to have been, acquired by an unauthorized person. [1]
BASEL II	Is recommendations on banking laws and regulations issued by the Basel Committee on Banking Supervision. Aplies to: Confidentiality and integrity of personal financial information stored by financial institutions. Availability of financial systems. Integrity of financial information as it is transmitted. Authentication and integrity of financial transactions [8].
Health Level Seven (HL7)	Provides regulations for the exchange of data among health care computer applications that eliminate or substantially reduce the custom interface programming and program maintenance that may otherwise be required [8].

Table 7.2: Accuracy of automatically generated datasets in two different areas of requirements engineering

Approach	Query	Correct
Big-Data	Query 1: Billing, Bill Calculation, Invoice Generation	90%
	Query 2: Balance Management, Credit Management, Account Management, Credit Card Processing	100%
	Query 3: Business Intelligence, SLA Management, Database Marketing	100%
	Query 4: Product Shipment, Shopping	100%
Web-Mining	Database Security HIPAA	63%

7.2 Usage Scenario#2: Classifying Functional Requirements.

Another area where automated techniques can be used is generating datasets for the problem of classifying/tracing functional requirements.

Problem: Frequently, the VSM [19] technique has been widely used for tracing functional requirements in source code. Conversely, there have been studies indicating the potential feasibility of using supervised machine learning techniques for tracing reoccurring functional requirements [5]. The most significant drawback with this approach is the difficulty of obtaining multiple samples of the same functional requirements or code snippets.

Feasibility Study: While using the Big-Data Analysis method, it was observed that, within the assembled ultra-large code repository, there are a significant number of software systems sourced from the same domain. As a result, the code snippets to implement functional requirements within that domain will also reoccur across these systems.

Therefore it is possible to collect datasets related to these implementations and use alternative supervised machine learning techniques in order to detect these types of requirements within the source code, or simply utilize such datasets for other purposes as necessary.

Table 7.2 shows the accuracy of the Big-Data Analysis method in establishing such datasets of code snippets that implement the functional requirements of OfBiz which is an ERP (Enterprise Resource Planning) software system. As shown, this was highly successful; in all cases, in fact, this approach successfully created datasets of relevant code snippets for implementing those requirements. The terms contained within the queries for retrieval of the implementation of functional requirements were directly extracted from an online documentation on a similar system [1]. .

[1] Please see terms in the figures: `http://www.1tech.eu/clients/casestudy_ventraq`

Chapter 8

Threats To Validity

There are four general classifications of threats to a research validity: *construct*, *internal*, *external*, and *statistical* validity. In this chapter, each type is discussed in how they might potentially affect this research and the efforts taken to mitigate their effects.

External validity evaluates the generalizability of the approach–how much the research is applicable outside of the exact area or population being studied. One of the primary threats in this area is in the construction of the study datasets.

The manually-compiled dataset included over 250 samples of tactic-related code. The task of locating and retrieving these code snippets was conducted primarily by two software architecture experts, with extensive experience in the area of software requirements, and was then reviewed by two additional experts. As mentioned, this was a very time-consuming task that was completed over the course of multiple months. This systematic process, especially the careful peer-review process, allows for high confidence in that each of the identified code snippets was indeed a representative of its relevant tactic.

In addition, each of the experiments conducted in this study were based on Java, C# and C code. A number of the identified key-terms were influenced by the constructs in these programming languages, such as calls for APIs that support a specific tactics implementation.

Moving onwards, the Hadoop and OfBiz case studies were structured with the intent of evaluating the impact of dataset size on the accuracy of tactic classification in a large and

realistic software system. It is therefore expected that these results will be representative of a typical software engineering environment, which then suggests that these results could then be generalized to a broader set of systems. Conversely, the majority of the identified keyterms are not language-specific. Regardless, the experimental results reported in this thesis will not be impacted by this particular issue.

Construct validity evaluates the degree to which the claims were correctly measured in the experiment(s). The n-fold cross-validation experiments that were conducted are a standard approach for the evaluation of results when the experiment structure renders the collection of larger amounts of data difficult. Additionally, to avoid the impact of adding additional variables, specifically that of dataset size on training-set quality, all the comparison experiments were conducted on a training set of equal size.

Internal validity is the measure of which a given study or research is consistent and avoids bias and error, allowing for a causal conclusion to be drawn. The greatest threat to this form of validity in the context of this thesis is that the search for specific tactics was limited by the preconceived notions of the experts, and that additional undiscovered tactics existed that used entirely different terminology–in essence, the risk is that the searchers found only those specific formulations of the tactics that they recognized, and overlooked ways of implementing the tactics that that they did not recognize, which would create a persistent bias in the research.

This risk was mitigated as best as possible via using search engines as the initial gathering mechanism and putting the results through a peer review. As multiple data collection mechanisms were used by the experts, the dataset is not dependent on a limited number of terms; in fact, in a number of cases, there was a large diversity in tactical terminologies.

In the case of the Hadoop project, we requested feedback from the Hadoop developers on the Hadoop open discussion forum.

Another threat in this category for this area of research is that the accuracy of the automated techniques can be dependent on the specific search query used in the study. To avoid this bias, the queries were preselected from textbook descriptions of the tactics.

Furthermore, for future work in this area, it is planned to run different experiments to identify the impact on the quality of the resulting datasets of domain-specific knowledge on the part of the person creating the query.

Statistical validity is essentially a measure of whether the papers research documents a real effect, or a statistical fluke, and whether any statistical analysis has been conducted correctly. In addressing this in the context of this thesis, appropriate statistical techniques were used for the analysis, with a specific focus on the reliability of the conclusions. Given the small sample sizes, two non-parametric tests were used. Both tests indicated that there was no statistically significant differences between the accuracies of the three training methods, although manual methods ranked as superior.

Chapter 9

Conclusions

This thesis presents three baseline techniques suitable for the creation of machine learning training sets, specifically in the area of training a classifier for the detection and classification of software architecture tactics, and the establishment of traceability links between software requirements, architectural tactics, and the source code.

The analysis of these three methods has shown that the automated techniques can generate useful training sets for this purpose. These datasets are statistically similar when compared to the resource-intensive and time-intensive standard method of compiling such datasets manually by experts in the field of computer software architecture. These proposed automation techniques appear to have potential use in multiple other areas of software engineering, such as the area of tracing architecturally significant requirements.

Furthermore, tangential research from the main point of the automated techniques indicates that there is a point of diminishing returns in the manual compilation of machine learning datasets. While this merits further study, it does mean that manually compiled, expert created data sets will not be rendered obsolete, as they are 100% accurate in their contents, while the automated techniques will always contain a degree of error. A possibility of a hybrid approach is also worthy of consideration.

However, at the moment, it is the conclusion of this report that the Big Data Analysis and Web Data Mining methods are potentially viable means by which to create the bootstrap training datasets for teaching algorithms to search, classify, and sort the means

and tactics by which software is structured. This has potential boons all on its own, as it would enable easier updating and patching of existing software, as the coders would be able to make better informed upgrades, alterations and choices in regards to the effects of their changes on the existing software. Other outcomes from this research are yet to be determined, but will be pursued.

9.1 Future Work

It is hoped that, as research in this area continues, that it will be possible to develop automated, unsupervised methods for creating these datasets that will possess similar or equal quality to the output of the manual compilations of the experts datasets. Such future work and long term goals will involve the investigation into these areas, as well as the potential practicality of applying these automated techniques into other areas of software engineering. Also in the realm of future work includes the potential for improved signal-to-noise ratios in the training sets, likely by means of engaging in a more fine-grained sampling of the source code files and web-pages. At this stage, the algorithms used are still primitive and have the potential for considerable refinement. Also worthy of consideration is analyzing the effect of the trace-users domain-specific knowledge on the quality of the resulting dataset–in effect, quantifying the effect of skill and experience on the output of these datasets.

Bibliography

[1] California Senate Bill SB 1386, Sept. 2002. http://www.leginfo.ca.gov/pub/13-14/bill/sen/sb_1351-1400/sb_1351_bill_20140221_introduced.pdf.

[2] Koders. http://www.koders.com.

[3] US Congress. Gramm-Leach-Bliley Act, Financial Privacy Rule. 15 USC 6801–6809, November 1999. http://www.law.cornell.edu/uscode/usc_sup_01_15_10_94_20_I.html.

[4] P. C. I. Council. Payment card industry (pci) data security standard. Available over the Internet - July 2010. https://www.pcisecuritystandards.org.

[5] Preethu Rose Anish, Balaji Balasubramaniam, Jane Cleland-Huang, Roel Wieringa, Maya Daneva, and Smita Ghaisas. Identifying architecturally significant functional requirements. In *Proceedings of the Fifth International Workshop on Twin Peaks of Requirements and Architecture*, TwinPeaks '15, pages 3–8, Piscataway, NJ, USA, 2015. IEEE Press.

[6] Felix Bachmann, Len Bass, and Mark Klein. *Deriving Architectural Tactics: A Step Toward Methodical Architectural Design*. Technical Report, Software Engineering Institute, 2003.

[7] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice, 3rd Edition*. Adison Wesley, 2003.

[8] George W. Beeler, Jr. and Dana Gardner. A requirements primer. *Queue*, 4(7):22–26, September 2006.

[9] Carla E. Brodley. Addressing the selective superiority problem: Automatic algorithm/model class selection. International Conference on Machine Learning, 1993.

[10] Horst Bunke et al. Generation of synthetic training data for an hmm-based handwriting recognition system. In *null*, page 618. IEEE, 2003.

[11] J. R. Cano, F. Herrera, and M. Lozano. Using evolutionary algorithms as instance selection for data reduction in kdd: An experimental study. *Trans. Evol. Comp*, 7(6):561–575, December 2003.

[12] Jane Cleland-Huang, Brian Berenbach, Stephen Clark, Raffaella Settimi, and Eli Romanova. Best practices for automated traceability. *Computer*, 40(6):27–35, 2007.

[13] Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *International Conference on Software Engineering (ICSE) (1)*, pages 155–164, 2010.

[14] Jane Cleland-Huang, Orlena Gotel, Jane Huffman Hayes, Patrick Mader, and Andrea Zisman. Software traceability: Trends and future directions. In *Proc. of the 36th International Conference on Software Engineering (ICSE), India*, 2014.

[15] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. Automated detection and classification of non-functional requirements. *Requir. Eng.*, 12(2):103–120, 2007.

[16] J.C.F. de Winter. Using the student's t-test with extremely small sample sizes. February 2014.

[17] Robert Dyer, Hridesh Rajan, Hoan Anh Nguyen, and Tien N. Nguyen. Mining billions of ast nodes to study actual and potential usage of java language features. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 779–790, New York, NY, USA, 2014. ACM.

[18] G. Gates. The reduced nearest neighbor rule (corresp.). *Information Theory, IEEE Transactions on*, 18(3):431–433, May 1972.

[19] M. Gethers, R. Oliveto, D. Poshyvanyk, and A.D. Lucia. On integrating orthogonal information retrieval methods to improve traceability recovery. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 133–142, Sept 2011.

[20] Marek Gibiec, Adam Czauderna, and Jane Cleland-Huang. Towards mining replacement queries for hard-to-retrieve traces. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 245–254, New York, NY, USA, 2010. ACM.

[21] Hongyu Guo and Herna L Viktor. Learning from imbalanced data sets with boosting and data generation: the databoost-im approach. *ACM SIGKDD Explorations Newsletter*, 6(1):30–39, 2004.

[22] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. Morgan Kaufmann, 1995.

[23] Gernot A. Liebchen and Martin Shepperd. Data sets and data quality in software engineering. In *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, PROMISE '08, pages 39–44, New York, NY, USA, 2008. ACM.

[24] Anas Mahmoud. An information theoretic approach for extracting and tracing non-functional requirements. In *Proc. RE*, pages 36–45. IEEE, 2015.

[25] Michael McCandless, Erik Hatcher, and Otis Gospodnetic. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., Greenwich, CT, USA, 2010.

[26] Jane Cleland-Huang Mehdi Mirakhorli. Detecting, tracing, and monitoring architectural tactics in code. *IEEE Trans. Software Eng.*, 2015.

[27] M. Mirakhorli. Preserving the quality of architectural decisions in source code, PhD Dissertation, DePaul University Library, 2014.

[28] Mehdi Mirakhorli and Jane Cleland-Huang. *Tracing Non-Functional Requirements.* In: Andrea Zisman, Jane Cleland-Huang and Olly Gotel. Software and Systems Traceability., Springer-Verlag., 2011.

[29] Mehdi Mirakhorli and Jane Cleland-Huang. Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 123–132, Washington, DC, USA, 2011. IEEE Computer Society.

[30] Mehdi Mirakhorli, Ahmed Fakhry, Artem Grechko, Mateusz Wieloch, and Jane Cleland-Huang. Archie: A tool for detecting, monitoring, and preserving architecturally significant code. In *CM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014)*, 2014.

[31] Mehdi Mirakhorli, Patrick Mäder, and Jane Cleland-Huang. Variability points and design pattern usage in architectural tactics. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 52:1–52:11. ACM, 2012.

[32] Mehdi Mirakhorli, Yonghee Shin, Jane Cleland-Huang, and Murat Cinar. A tactic centric approach for automating traceability of quality concerns. In *International Conference on Software Engineering, ICSE (1)*, 2012.

[33] Maria Luiza C. Passini, Katiusca B. Estbanez, Grazziela P. Figueredo, and Nelson F. F. Ebecken. A strategy for training set selection in text classification problems. *(IJACSA) International Journal of Advanced Computer Science and Applications*, 4(6):54–60, 2013.

[34] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[35] David B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 293–301. Morgan Kaufmann, 1994.

[36] Irvine University of California. The sourcerer project. sourcerer.ics.uci.edu.

[37] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-basedlearning algorithms. *Mach. Learn.*, 38(3):257–286, March 2000.

[38] Roozbeh Zarei, Alireza Monemi, and Muhammad Nadzir Marsono. Automated dataset generation for training peer-to-peer machine learning classifiers. *Journal of Network and Systems Management*, 23(1):89–110, 2015.

[39] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. Patterns of folder use and project popularity: A case study of github repositories. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 30:1–30:4, 2014.